APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: REGULATING FILE SYSTEM DEVICE ACCESS

APPLICANT: JOHN C. WEAST

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL688324186US

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

November 16, 2001
Date of Deposit


Signature

Gabe Lewis
Typed or Printed Name of Person Signing Certificate

REGULATING FILE SYSTEM DEVICE ACCESS

Background

[0001] The present application describes systems and techniques for regulating access to file system devices in a computer system, for example, a mobile computing platform such as a laptop computer or a hand-held computer.

[0002] A typical laptop computer includes one or more file system devices through which files such as application files, data files, operating system files, and the like can be accessed - for example, written or read. Such file system devices include non-volatile data storage devices such as disk drives (e.g., hard disk, floppy disk, zip drive, CD or DVD drive, or non-volatile memory components such as flash cards) as well as other devices such as network interface adapters, or any other network access devices, which typically do not themselves store data but rather serve as a conduit through which file write and/or read requests are communicated to a data storage resource on a network.

[0003] Many users rely on their mobile computers for performing computing tasks, for example, running application programs such as word processors, spread-sheet programs, graphic presentation programs and the like, in the absence of the wall-

socket AC power sources that are present in virtually all offices and homes. Consequently, a mobile user by necessity often relies on battery power for operating his or her mobile computer. Many users discover, however, that the battery life of their mobile computer is less than they would optimally desire. A mobile user on a long-distance flight may, for example, experience a lapse in battery life, and thus be forced to cease using the computer, while only half-way through the flight. This may be particularly true if the user is operating devices in the mobile computer such as disk drives, which typically exhibit a relatively high level of power consumption compared to other devices in the system.

[0004] In an effort to extend battery lifetime, several computer and/or operating system manufactures have developed utilities that enable users to regulate the behavior of their mobile computers while operating under battery power. These utilities can control properties such as screen brightness levels and the length of a hard drive time-out period - that is, the time duration of hard drive inactivity that may occur before the hard drive is forced to power down or otherwise assume a reduced power state.

[0005] In addition to the manufacturer-specific implementations of power saving utilities, various organizations

have published specifications relating to power management. For example, Intel Corp. and Microsoft Corp. jointly published a power management specification entitled "Advanced Power Management (APM): BIOS Interface Specification," Rev. 1.2, Feb. 1996. Another specification, "Advanced Configuration and Power Interface Specification," Rev. 2.0, July 27, 2000, was published jointly by Compaq Computer Corp., Intel Corp., Microsoft Corp., Phoenix Technologies Ltd. and Toshiba Corp.

Drawing Descriptions

[0006] Fig. 1 is a block diagram of an implementation for regulating access to a file system device.

[0007] Fig. 2 is a flowchart of a process for selectively buffering file write requests.

[0008] Fig. 3 is a flowchart of a process for selectively committing buffered write operations to a file system device such as a disk drive or network adapter.

[0009] Fig. 4 is a flowchart of a process for regulating file read requests from a file system device such as a disk drive or network adapter.

[0010] Details of one or more embodiments are set forth in the accompanying drawings and the description below. Other

features and advantages will be apparent from the description and drawings, and from the claims.

Detailed Description

[0011] The present inventor recognized that conventional power management techniques tended to provide inadequate solutions for conserving or otherwise managing battery power in mobile computing platforms. In particular, the present inventor recognized that access to file system devices while on battery power appeared to be performed by applications and/or the operating system in a sub-optimal manner. For example, when operating a mobile computer system under battery power, the hard disk, which previously had timed-out and deactivated to conserve power, may unexpectedly spin up and/or go through a series of spin up / spin down sequences. Either of these occurrences may waste power and shorten battery lifetime. Accordingly, systems and techniques as described herein have been developed that provide a robust scheme for flexibly and intelligently controlling access to file system devices, for example, depending on the power state of the device to be accessed. As a result, the battery lifetime in a mobile computing platform may be extended while minimizing adverse effects on performance and/or functionality.

[0012] In a typical conventional system, file system requests (e.g., to read or write data) are handled by a file system driver that fulfills the requests by accessing the file system device (e.g., a disk drive, a network adapter, etc.) to read and/or write data as appropriate. If the computer system is operating on battery power and the device has been powered down or otherwise deactivated (e.g., due to the lapse of a time-out condition), typically the device first must be powered up or otherwise activated before the file system request can be fulfilled.

[0013] Fig. 1 is a block diagram of an implementation for regulating access to a file system device in a manner that may help to minimize unnecessary device access operations and/or unnecessary device activation-deactivation operations, and thus may help to conserve battery power in a mobile computing platform. As shown in Fig. 1, file system requests 100 ordinarily handled by a file system driver (FSD) 104 may be intercepted and handled in the first instance by an intermediate FSD 102, which may be implemented as a software component running in conjunction with the operating system. Alternatively, the functionality of the intermediate FSD could be implemented in a variety of ways - e.g., as a native functionality of the operating system. Generally speaking,

implementation of an intermediate FSD or its functional equivalent is platform specific. Regulating access to a file system device also could be implemented as a native functionality of the storage device itself, for example, by including appropriate software, firmware and/or hardware in a storage device such as a hard-drive.

[0014] By intercepting file system requests 100, the intermediate FSD 102 may intelligently and flexibly fulfill the requests in a manner that minimizes device access operations and/or device activation-deactivation operations, thereby potentially conserving power or other resources. For example, the intermediate FSD 102 can intercept file system write requests and selectively buffer them to physical memory 110 (e.g., one or more semiconductor-based memory components) until a predetermined condition is detected or occurs. Such predetermined conditions may include detecting that the write buffer has become full, that a certain amount of time has passed, that the battery power is approaching or has reached a specified threshold level, that the computer system is being turned off or put in a standby state, and/or that a user, a process or the operating system has explicitly requested that the write buffer contents be committed to non-volatile storage by the device directly (e.g., in the case of the device being a

disk drive) or indirectly through the device (e.g., in the case of the device being a network adapter).

[0015] When such a predetermined condition is detected, the buffered write requests may be read from physical memory 110 and written to the appropriate device 106. In some situations, the buffer may include two or more write requests that seek to modify the exact same memory locations in the storage device. For example, a first write request may seek to modify a storage location XX to hold the data value 100 and a second, subsequent (i.e., later in time) write request may seek to modify that exact same memory location XX to hold the data value 101. In this example, buffering and committing all of the write requests in the order received would produce correct results but would be inefficient in that location XX first would be modified to hold 100 but then would be overwritten almost immediately to hold 101.

[0016] Accordingly, buffering write requests can be performed intelligently, for example, by deleting an earlier write request from the buffer when a subsequent write request to the same storage location is intercepted and buffered. As a result, the size of the buffer remains smaller than it otherwise would if all write requests to identical storage locations were buffered. Moreover, committing the buffered write requests can be

performed more quickly and efficiently since potentially fewer write requests may need to be performed.

[0017] Selectively buffering write requests can be performed on a process-specific or case-by-case basis. For example, certain applications (e.g., application programs such as Microsoft Word or Microsoft PowerPoint) could selectively register with the intermediate FSD 102 (e.g., by reference to a specific application program interface (API) call associated with the intermediate FSD 102) to identify themselves as participating in the selective buffering scheme. Such applications could register, for example, such that only select file types are subject to selective buffering of write requests or all write requests from that application type are subject to selective buffering.

[0018] Other processes (e.g., vital system utilities or the operating system itself) could, depending on the user's and/or designer's preferences, choose to be or deemed to be exempt from the selective buffering scheme altogether, in which case file system requests 100 may be passed on to and handled by the FSD 104 directly in a conventional manner. Further, a process could register to participate in the intermediate FSD buffering scheme for some file system requests but not others depending, for example, on state or context.

[0019] Optionally, a user customization interface 108 (e.g., a Ring 3 application that communicates with a user through a graphical user interface (GUI)) could be provided to enable users of the computer system to specify one or more parameters or characteristics of the selective buffering scheme described here. Such parameters may include items such as which applications or processes are to participate in, or be excluded from, the buffering scheme, buffer size, time-out periods, and/or specific conditions under which the selective buffering should be employed or bypassed. For example, the user customization interface 108 may enable the user to specify circumstances under which a file write request will not be buffered but rather will force a device access to commit the data to non-volatile storage. These conditions may include, for example, an occurrence of a user's explicitly choosing the "save" command from the application's menu structure, a specified percentage of the document being changed, a specified volume of data being buffered, a specified number of writes being buffered, the battery's power level reaching a specified threshold, and/or the passage of a specified amount of time since the last committing of the buffer's contents to non-volatile storage. In general, the user customization interface 108 may allow a user to override any settings, defaults or other

conditions that may have come about as a result of interactions among the various software components in the system (e.g., operating system, application programs, utilities, etc.).

[0020] The implementation of Fig. 1 may be used alternatively, or in addition, for selectively caching data to fulfill file system read requests while operating, for example, in a limited power state (e.g., while the laptop computer is operating under battery power). As a result, device access operations and/or device activation-deactivation operations typically caused by file system read requests may be minimized, thereby potentially conserving power or other resources. For example, if the device 106 is deactivated, a file system read request 100 directed at the device 106 may be intercepted by the intermediate FSD and fulfilled by accessing physical memory 110 if the data already has been cached. Alternatively, if the requested data has not yet been read from the device 106 cached to physical memory 110, then the intermediate FSD 102 may cause the device to become active (e.g., powered up) and read not only the requested data to fulfill the read request but also a superset of the data logically related to the requested data. This superset may be either the entire file or less than the entire file, depending on usage, context and computing

environment conditions (e.g., physical memory size, device access time, etc.).

[0021] For example, if a file system request 100 seeking only a portion of a file (e.g., an application program such as MS PowerPoint or a data file such as a user-defined presentation) is intercepted, the intermediate FSD 102 may read the entire file from the device 106 and cache the file to physical memory 110. Subsequent file system read requests 100 for other portions of the file then may be fulfilled by reference to the physical memory 110, and the device 106 can be deactivated either as a result of a time-out or in response to a specific command from the intermediate FSD 102 to do so. In either case, to ensure maximum power savings benefit, the device typically should be kept deactivated for as long as feasible to do so.

[0022] Certain processes or file system requests may choose to be, or deemed to be, exempt from the selective caching scheme. Moreover, the user customization interface 108 may enable users to specify parameters relating to such selective caching such as such as which applications or processes are to participate in the caching scheme, cache size, time-out periods, and the like.

[0023] By selectively buffering file system write requests and/or read requests in the manners described, the

implementation of Fig. 1 may provide several advantages. For example, non-critical disk drive writes, which would otherwise cause a deactivated disk drive to power up, may be buffered in physical memory for as long as feasible - that is, until a condition arose that required the buffered writes to be committed to disk. In the interim, if a process or application requests to read a file or sector that was previously written to the buffer, the data can be returned from physical memory. On the other hand, file system requests deemed to be critical (e.g., from a system driver) could be exempted from the buffering scheme and allowed to proceed in usual fashion in order to ensure the system's continued health.

[0024] In addition, disk drive spin-up / spin-down sequences that may be experienced when reading different portions of a file may be minimized by reading the entire file into physical memory cache in response to the first such read request and then fulfilling subsequent read requests from the cached file. In general, a mobile computer operating under the implementation of Fig. 1 may minimize the number of device activation-deactivation sequences thereby extending battery lifetime. Reducing the number of activation-deactivation sequences also may provide other benefits such as extending a disk drive's time between failures, and reducing data access latency.

[0025] Fig. 2 is a flowchart of a process 200 for selectively buffering file write requests. The process 200 could be executed by a mobile computing platform whenever it is determined that a limited power condition exists, for example, when operating under battery power.

[0026] First, a file write request seeking to write data to a device is intercepted (e.g., in the case of an intermediate FSD implementation) or otherwise directed to the appropriate process (e.g., in the case of an enhanced operating system functionality) (202). Next, the process 200 determines whether the device under consideration already is activated, for example, powered-up (204). If so, the device may be accessed in the usual manner to fulfill the write request (206). If not, the process 200 determines whether the file type of the file write request is registered with the process 200 as participating in the selective buffering scheme (208). If not, the device is activated (e.g., powered up) (210) and accessed (206) to fulfill the write request.

[0027] Next, the process 200 determines whether one or more other conditions may exist that require the device to be accessed to fulfill the write request (212). As noted previously, examples of such conditions may include detecting that the write buffer has become full, that a certain amount of

time has passed, that the battery power is approaching or has reached a specified threshold level, that the computer system is being turned off or put in a standby state, and/or that a user, a process or the operating system has explicitly requested that the write buffer contents be committed to non-volatile storage. Committing the buffer contents to non-volatile storage in this manner is not limited to being performed only as a part of the process 200 but rather may be performed whenever one or more predetermined conditions are detected.

[0028] If one or more of these other conditions is detected, the device is powered up (210) and accessed (206) to fulfill the write request. Otherwise, the file write request is buffered to physical memory (214) and resides there until a predetermined condition is detected, at which time the buffered write requests are committed to non-volatile storage. While residing in physical memory, buffered write requests may be accessed, for example, to fulfill read requests seeking access to the buffered data.

[0029] The techniques and systems described here also may be used with file systems that have a native capability for buffering or caching file system requests. In Microsoft Windows NT File System (NTFS), for example, file system write requests can be buffered and then periodically committed to disk. The

buffering performed by NTFS, however, is performed without regard to a power state of the disk. Accordingly, NTFS could have a number of writes buffered, then due to inactivity that exceeds the disk time-out period, the disk may be powered down only to have to be powered up again, potentially seconds or minutes later, as soon as it is time again for NTFS to commit the buffered writes to disk. When operating under battery power or otherwise under a limited power condition, such potentially unnecessary disk power-up / power-down sequences may waste power and reduce battery life.

[0030] Fig. 3 is a flowchart of a process 300 for selectively committing buffered write operations to a file system device, such as a disk drive or network adapter, in a manner that may take into account power state. As shown in Fig. 3, the process 300 waits until the device's time-out period is to expire (302), indicating that the device is to be powered-down or otherwise deactivated according to parameter settings in the system's power manager. Next, the process 300 determines whether the system is operating under limited power conditions, such as operating on battery power (304). If not, the device is allowed to deactivate in the usual manner. If so, the process 300 determines whether any write operations have been buffered to physical memory since the last commit operation (308). If not,

it means that there is nothing in the write buffer to commit to disk, so the disk is allowed to deactivate (306). On the other hand, if it is determined that one or more write operations have been buffered, then the buffered writes are retrieved from the buffer and committed to the device (310) before the device is allowed to deactivate (306).

[0031] The processes illustrated in Figs. 2 and 3 could be modified in various manners while still achieving advantageous results. In particular, the order of operations could be altered and/or additional or different operations could be performed instead. For example, the process of selectively buffering write requests to physical memory could be configured such that any buffered write requests are committed to non-volatile storage at a convenient or otherwise opportune time. If, for example, the device becomes activated as a result of a critical device access request initiated, e.g., by a system driver or the operating system, then the selective buffering process could take advantage of the device's active state by writing the buffered files to the device at that time rather than waiting for the occurrence of another condition.

[0032] Fig. 4 is a flowchart of a process 400 for regulating file read requests from a file system device such as a disk drive or network adapter. The process 400 may be used for

implementing a selectively file caching technique that may reduce a number of device activation-deactivation sequences while operating in a limited power state.

[0033] First, the process 400 intercepts a request to read a file segment from a specified device (402), for example, a portion of an application program corresponding to a function that is being invoked, or a portion of a content file that is to be displayed. Next, the process 400 determines whether the entire file already has been read from disk and loaded into physical memory (404). If not, the process 400 next determines whether the device (e.g., a disk drive) is powered-up or otherwise activated (408) and, if not, powers-up the device (410). Next, the process 400 determines whether the file type of the requested file is registered (412) - e.g., designated by the user or otherwise as participating in the selective caching scheme. A particular file type may be deemed to be registered either if the file type is registered as an independent entity, or if the file system request came from an application for which all file system requests are to be treated as subject to the selective file caching technique.

[0034] If the file type is not registered, then the requested file segment is read from the device in the usual manner (414). On the other hand, if the file type is registered, then the

process 400 reads the requested file segment from the device (416) and returns it to the requesting process or application. Then, the remainder of the file is read from the device (418) and the entire file is loaded into a portion of physical memory used the selective caching process (420). In the implementation shown in Fig. 4, the requested file segment is read first and returned before the remainder of the file is read and stored in order to minimize undue delays in returning the requested memory portion. However, other implementations could function differently, for example, the entire file contents could be read from the device first and then the requested file portion could be returned.

[0035] After the entire file has been read and stored in physical memory, the process 400 can fulfill subsequent file system read requests relating to that file from memory without having to access the device. Consequently, the process 400 optionally can deactivate the device immediately after reading the entire file from the device. In any event, the next time that a file system read request is intercepted seeking a portion of the cached file (402) and the process 400 confirms that the entire file has been loaded into physical memory (404), the process 400 can return the requested file segment from memory (406) without having to activate or access the device.

[0036] Variations of, or modifications to, the process 400 may be implemented advantageously depending on context and usage. For example, rather than reading in the remainder of the entire file in (418), the process 400 may read in less than the entire file, e.g., a superset of the requested file portion but, at the same time, a subset of the entire file. This superset may be logically related to the portion of the file read to fulfill the read request. This practice may be useful where a file, such as a relational database file, is too large or otherwise impractical to be read entirely into available physical memory.

[0037] For example, assume that the intercepted read request (402) is a request to read a row from a table in a 300 MegaByte (MB) relational database file. In that case, the entire 300 MB file generally is too large to be read entirely and stored in a typical computer system's physical memory. Instead, the process 400 may read in not only the requested row from the database file, but also an additional portion of the database file that is likely to be accessed by future read requests. The amount of the file to be read into physical memory may depend on one or more factors, including available physical memory space, user defined parameters, file size and the like. The additional portions of the file to be read into physical memory may be

chosen intelligently, for example, using predictive caching techniques or the like. In the database file example, the additional portions to be read (i.e., the superset of the requested file portion) may include adjacent rows, the entire table, and/or other logically related information fields. In this regard, a file access monitor process could be implemented that keeps track of, and identifies trends relating to, which files, and which portions of files, have been accessed recently, e.g., for the last 3-4 days or so. The intelligent decisions as to which file portions to be read into physical memory by process 400 could be based on the information collected and maintained by the file access monitor. Alternatively, or in addition, a secondary memory paging manager (discussed below) could use information from the file access monitor to make intelligent decisions about which page or pages are to be removed from physical memory, e.g., in order to accommodate additional portions of a file read in response to an intercepted read request.

[0038] The process 400 may provide several advantages. For example, in a conventional file system implementation, application files, and/or content files, typically may be read from disk in a piecemeal manner - for example, a series of disk read operations may be performed to read in different portions

of the application file corresponding to different application functions as they are invoked. This piecemeal approach, as a result, tends to increase the likelihood that a user will experience multiple disk activation-deactivation sequences, each of which may contribute to decreased battery life. Assume, for example, that a user is giving a PowerPoint slide presentation while on battery power, the user spends five minutes speaking about each slide, and the disk time-out period is set to three minutes in an effort to conserve battery power. In a conventional file system implementation, each slide change under this scenario typically would result in a separate disk deactivation-activation sequence. When several such deactivation-activation sequences occur during the course of a multiple slide presentation, the cumulative effect may be to reduce battery lifetime considerably.

[0039] In contrast, the selective caching technique illustrated in Fig. 4 if used in this scenario typically could require only a single disk deactivation-activation sequence and thus may result in a significant power savings. Many or possibly even most of the computer systems being sold and/or currently in place have sufficient physical memory to load and store entire application files. A typical computer system, for example, having 256 MB of physical memory might have seven

different applications open concurrently and still use only about 140 MB of actual memory (physical plus virtual), leaving about 116 MB of physical memory available for caching applications or other files. Of course, these numbers may vary considerably depending on various factors such as the computer's memory configuration, operating system, processor speed, etc.

[0040] Once the entire file, or selected portions thereof, are cached in physical memory, the disk or other device can be deactivated and typically need not be activated again while using the cached application. Under this scheme, further read requests may be intercepted and fulfilled from physical memory thereby leaving the disk spun down to conserve power.

[0041] Additional or different features or functionalities may be implemented. For example, statistics about access to specific system calls and/or libraries could be gathered and used to selectively cache frequently accessed resources to physical memory and/or to make intelligent decisions about which additional portions of a file are to be cached in response to an intercepted file request. For example, on a Microsoft Windows platform, if a specific library file residing on disk, e.g., network.dll, is accessed frequently by one or more processes, whether an application, a system driver or another type of process, then the entire file (or selected portions thereof)

could be cached to physical memory thereby minimizing, or eliminating, the need to subsequently activate the disk to access this file.

[0042] Alternatively, or in addition, the various applications registered to participate in the selective buffering / caching schemes could be accorded (e.g., by the user through the user customization interface) relative priorities that could be used in making buffering / caching decisions. For example, a secondary memory paging manager could be implemented that overrides memory paging decisions (e.g., deciding which page of memory to load into and/or remove from physical memory) depending on the relative caching priority of currently executing applications. The secondary memory paging manager may ensure that, for file system read requests initiated by registered file types and/or application types, the resulting memory pages read from disk are stored in physical memory rather than in virtual memory (which typically would be back on the disk). In contrast, the secondary memory paging manager may choose to store read memory pages resulting from unregistered file types or applications in virtual memory. Further, when physical memory is full or nearly so, the secondary paging manager could use relative (e.g., user-specified) caching priorities of registered file types / applications to decide

which corresponding memory pages are to be stored in physical memory and which may be relegated to virtual memory.

[0043] In addition, the secondary memory paging manager may override or otherwise influence physical memory "victim" decisions - that is, decisions as to which page or pages to be removed from physical memory when the need arises. MS PowerPoint, for example, could be assigned a higher priority because it may be more likely to require frequent disk accesses to read in application instructions or file data. In this example accordingly if PowerPoint is running as the same time as a lower priority application, the secondary memory paging manager may influence physical memory victim decisions to keep the higher priority PowerPoint memory page reads in physical memory for as long as feasible, while allowing the memory pages associated with lower priority applications to be removed from physical memory. More generally, if multiple applications are registered to participate and are running concurrently and not enough physical memory exists to cache all of them, their relative priorities could be used to decide which are cached and which are not.

[0044] The above systems and techniques also may find application in systems other than mobile computing platforms. For example, by minimizing the occurrence of device activation /

deactivation sequences, the techniques described here may increase the hardware lifetime and/or mean time between failures of devices residing in either mobile or stationary systems. In addition, in stationary systems such as disk farms and the like that that have relatively large power requirements, these techniques may be used to reduce the cumulative amount of time that the disks are activated, thereby potentially realizing significant power and cost savings.

[0045] Various implementations of the systems and techniques described here may be realized in computer hardware, firmware, software, or combinations thereof, or in digital electronic circuitry, integrated circuitry, field programmable gate arrays (FGPAs), specially designed ASICs (application specific integrated circuits).

[0046] Other embodiments may be within the scope of the following claims.